A LINEAR LOGIC VIEW OF GAMMA STYLE COMPUTATIONS AS PROOF SEARCHES

PAOLA BRUSCOLI
Università di Ancona, Istituto di Informatica, via Brecce Bianche, 60131 Ancona, Italy and GMD-FIRST, Rudower Chaussee 5, 12489 Berlin-Adlershof, Germany

ALESSIO GUGLIELMI
Università di Pisa, Dipartimento di Informatica, Corso Italia 40, 56125 Pisa, Italy

ABSTRACT *Using the methodology of abstract logic programming in linear logic, we establish a correct and complete translation between the language* NABLA *and first order linear logic.* NABLA *is a modification of the coordination language* GAMMA *with parallel and sequential composition.* NABLA*, without modifying* GAMMA *basic computational model, is amenable to this kind of analysis, at the price of a weaker expressive power. The translation is correct and complete in the sense that we establish a two way correspondence between computations in* NABLA *and the search for proofs in a suitable fragment of first order linear logic. Moreover, the translation is not an encoding, meaning that to the algebraic structure of* NABLA *programs is assigned logical meaning through a non-trivial use of linear logic connectives, as opposed to merely reflecting their operational behavior through a simulation into terms of the logic. In this way we hope that the connection established between the two formalisms can compensate for the diminished expressive power of* NABLA *with a powerful analysis tool, which could lead both to theoretical and practical improvements in semantic foundations of* GAMMA*-style languages and in the design of efficient implementations of their interpreters. The main difficulty has been to deal with sequential composition of programs, and to smoothly integrate its logical treatment in a recursive framework. An intermediate step is the definition of the language* SMR*, by which it is possible to specify in a very intuitive way* NABLA *operational semantics, and to prove that this specification is actually equivalent to the SOS-style one derived from* GAMMA *semantics.*

1 Introduction

The GAMMA coordination language [1] gave rise to the so-called "chemical metaphor." Computations are seen as the concurrent application of "chemical reactions" to an unstructured solution of "molecules," where reactions may be thought of as non-deterministic programs which act on data carried by molecules. The GAMMA language, which relies essentially on multiset rewriting, naturally yields a programming methodology that does not impose artificial sequentiality on the execution. A basic GAMMA program is a set of rules B which transform a multiset M until no rules are applicable; when this happens the computation stops.

GAMMA has been later enhanced with two composition operators on programs [2]. $Q \mid Q'$ and $Q \,;\, Q'$ stand, respectively, for the parallel and sequential composition of programs. This paradigm has been proven fruitful and a number of investigations has been devoted to it (among them see [3, 4] for issues regarding this work). In this paper we study the problem of specifying a "chemical," GAMMA-like language in first order linear logic.

Linear logic [5] is a powerful and elegant framework in which many aspects of concurrency, parallelism and synchronization find a natural interpretation. The difficulties of dealing with these issues within classical logic are overcome by the linear logic approach, mainly thanks to the "resource-orientation" of its multiplicative fragment. This roughly amounts to a good treatment of logical formulas as processes, or agents, in a distributed environment [6, 7]. The richness of the calculus and the deep symmetries of its proof theory make it an ideal instrument for purposes such as language design and specification, operational semantics, and it is certainly an interesting starting point for denotational semantics investigations. We are interested here in the "(cut-free) proof search as computation" paradigm, as opposed to the "cut-elimination as computation" one.

We argue in this paper that GAMMA with the parallel and sequential composition operators is not exactly specifiable in linear logic without resorting to some encoding of its structure into terms of the logic. This is something in general we can always do, provided formalisms are Turing equivalent. Of course, it would be much more interesting if the structural content of programs and computations were reflected into the connectives of the logic, instead of recurring to trickeries with terms. In this way one can hope to use logic in a non-trivial way, and not as a mere elegant suitcase.

Thus, in this paper we define a language, NABLA, which is weaker than GAMMA in its expressive power, but which is amenable to a correct and complete specification in a fragment of first order linear logic. From this analysis it should be evident which problems one encounters with GAMMA and why some trade between expressive power and good logical specification must be made. A possible outcome of this investigation could be the need to design some new logic more suitable to the kind of problems we encounter, and this work can give some hints about what is needed and what is not.

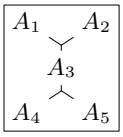
In fact, to specify NABLA into linear logic, a long journey has to be made. While the parallel execution of two programs $Q \mid Q'$ finds a natural understanding as $Q \parr Q'$ (or $Q \otimes Q'$ in a symmetrical interpretation), the same cannot be said for their *sequential* composition $Q \,;\, Q'$. We can naively achieve sequential composition in an indirect way, through backchaining. This is not

satisfactory for at least two reasons: because it is an unnatural form of encoding, and because backchaining is most naturally thought of, and dealt with, as a non-deterministic tool, while sequential composition is deterministic. A major problem one encounters when trying to express sequentialization is having to make use of "continuations," which are, in our opinion, a concept too distant from a clean, declarative, logical understanding of the subject.

In this paper we deal with sequentiality in a way which certainly does not have the flavor of continuations. Sequentialization is achieved in linear logic by a controlled form of backchaining, whose non-determinism is eliminated by the linearity of the calculus (linear implication) and a declarative way of producing unique identifiers (universal quantification). In our case these two mechanisms, together with the usual $\parr$ one, are embodied in a translation with a clear declarative meaning.

We briefly present the language SMR (Sequential Multiset Rewriting) [8] and give a translation of it into first order linear logic which is both correct and complete, thus fully relating the two formalisms. Computing in SMR is in the logic programming style: a goal of first order atoms (*agents*) has to be reduced to empty through backchaining by clauses, thus producing a binding for variables. Goals are obtained from agents by freely composing with the two connectives $\diamond$ (*parallel*) and $\triangleleft$ (*sequential*). Every top agent, *i.e.* every agent not preceded by other agents, can give birth to a new subgoal. The declarative meaning of $A \diamond A'$ is that we want to solve problems (to prove) A *and* A'; the meaning of $A \triangleleft A'$ is that we want to solve A *and then* A'. The simplest way to introduce synchronization in this framework is having clauses of the form $A_1, \ldots, A_h \multimap G_1, \ldots, G_h$. They state the simultaneous replacement of top agents $A_1, \ldots, A_h$ with goals $G_1, \ldots, G_h$, respectively. This framework has been studied by Monteiro in a more complex formal system called "distributed logic" [9, 10].

It is natural to associate hypergraphs to goals: nodes are agents and hyperarcs express the immediate sequentiality relationship among agents. Thus the hypergraph relative to $G = (A_1 \diamond A_2) \triangleleft A_3 \triangleleft (A_4 \diamond A_5)$ is



.

Let us associate to every agent A_i the empty agent $\circ_i$, whose declarative meaning is "agent in position i has been solved." A natural description in linear

logic of the goal G is given by the formula

$$\Big(\big(A_3 \multimap (\circ_1 \parr \circ_2)\big) \otimes \big((A_4 \parr A_5) \multimap \circ_3\big) \otimes (\circ_4 \parr \circ_5)\Big) \multimap (A_1 \parr A_2).$$

Here indices of agents have to be thought of as unique identifiers of the position of the agent in the goal. Now we need something more: since subgoals appear during the computation as an effect of resolutions, we need a mechanism to "localize" goal descriptions in linear logic, so as to fit them to the contingent goal dynamically. Again, a natural way to do that is describing G as

$$\forall i_1 i_2 i_3 i_4 i_5 : \Big(\Big(\big(A_{i_3} \multimap (\circ_{i_1} \parr \circ_{i_2})\big) \otimes \big((A_{i_4} \parr A_{i_5}) \multimap \circ_{i_3}\big) \otimes (\circ_{i_4} \parr \circ_{i_5})\Big) \multimap (A_{i_1} \parr A_{i_2})\Big).$$

We do not really need $\otimes$ since $(A_1 \otimes \cdots \otimes A_h) \multimap A \equiv A_1 \multimap \cdots \multimap A_h \multimap A$. It turns out that this very simple-minded idea actually works. Moreover, the $\circ$ goal behaves as a unity for $\diamond$ and $\triangleleft$, as *true* does for *and* in classical logic.

SMR is a plain generalization of Horn clauses logic programming, using $\diamond$ instead of $\wedge$. As a matter of fact, considering clauses of the form $A \mathrel{\Leftarrow} A_1 \triangleleft \cdots \triangleleft A_h$, we grasp PROLOG's left-to-right selection rule, and of course many more selection rules and much greater control over the order of execution of goals are possible.

In order to link SMR to linear logic we use a fragment of FORUM [11], which is a presentation of linear logic from an abstract logic programming perspective [12]. Its choice is rewarding because FORUM puts under control a large amount of the non-determinism of linear logic, which is something in the direction we are pursuing.

Summarizing, the link between NABLA and linear logic is established in three steps, which in turn preserve correctness and completeness:

1) FORUM $\leftrightarrow$ linear logic.
2) SMR $\leftrightarrow$ FORUM.
3) NABLA $\leftrightarrow$ SMR: the final step amounts to a specification of NABLA by SMR and proving its correspondence to the modified SOS-style operational semantics of GAMMA.

Sect. 2 is devoted to preliminaries and FORUM, in sect. 3 we present SMR and its operational semantics; then, in sect. 4, the translation into FORUM is shown and correctness and completeness are stated. In sect. 5 NABLA is defined, and in sect. 6 a translation of it into SMR is given, together with the proof of its correctness and completeness.

2 Basic Notions and Preliminaries

The first subsection fixes the notation for some usual preliminaries. In the second one a brief exposition of the fragment of FORUM we are interested in is given.

2.1 Notation and Basic Syntax

Let S and S' be sets: Then $S \setminus S'$ stands for their difference $\{ s \in S \mid s \notin S' \}$; $\mathsf{P}_\mathsf{f}(S)$ stands for the set of finite subsets of S; if h is a positive integer, S^h stands for the set $\underbrace{S \times \cdots \times S}_{h}$.

N is the set of the natural numbers $\{0, 1, 2, \dots\}$. Given $h \in \mathsf{N}$, indicate with N_h the set $\{h, h+1, h+2, \dots\}$; given $k \in \mathsf{N}$, indicate with N_h^k the set $\mathsf{N}_h \setminus \mathsf{N}_{k+1}$. Given $h, k \in \mathsf{N}$, if $h \leqslant k$ then $e|_h^k$ stands for "$e_h, \dots, e_k$"; if $h > k$ then $e|_h^k$ and $(e|_h^k)$ stand for the empty object "".

Given a set S, indicate with S^+ the set $\bigcup_{i \in \mathsf{N}_1} S^i$ and with $S^\star$ the set $S^+ \cup \{\epsilon_S\}$, where $\epsilon_S \notin S^+$. ϵ_S is the *empty sequence* (*of* S) and at times we shall write ϵ or nothing instead of ϵ_S. Let $S^0 = \{\epsilon_S\}$.
On sequences is defined a *concatenation* operator $\|$, with unity ϵ.
Given the (possibly infinite) sequence $Q = (s_1, s_2, \dots)$ and given $f\colon S \to S'$, if $s_1, s_2, \dots \in S$ define $f(Q)$ as $(f(s_1), f(s_2), \dots)$.

Multisets are sequences of elements where the order among elements does not count, or, alternatively, are sets in which the arities of elements count.

A *multiset* whose elements are in set S is a function $M\colon S \to \mathsf{N}$. When explicitly provided, a multiset M is denoted as $\{m|_1^h\}_+$, where every element of multiplicity $k > 0$ appears exactly k times. $\Subset$ denotes the submultiset relation, $\uplus$ is multiset union and $\setminus_+$ is multiset difference. $\mathsf{P}_+(S)$ denotes the set of multisets with elements in S.

In the rest of the paper, we shall frequently adopt the following convention: cursive roman letters (as P, the set of programs) denote sets whose generic elements shall be denoted by the corresponding italic letter (as P, a generic program). Therefore we shall often consider implicit such statements as $P \in \mathsf{P}$. Every newly introduced syntactic symbol or class of symbols shall be considered different or disjoint from the already introduced ones.

f, x and p are respectively the sets of *functions*, *variables* and *predicates*. They are denumerable and the functions $\mathrm{ar}\colon \mathsf{f} \to \mathsf{N}$ and $\mathrm{ar}\colon \mathsf{p} \to \mathsf{N}$ (*arities*) are

defined. 0-arity functions are called *constants*.
The set of *terms* t is the least set such that:

1) $\mathsf{x} \subset \mathsf{t}$.

2) If $f \in \mathsf{f}$ and $t_1, \ldots, t_{\mathrm{ar}\, f} \in \mathsf{t}$ then $f(t|_1^{\mathrm{ar}\, f}) \in \mathsf{t}$.

Let $\mathsf{A} = \{\, p(t|_1^{\mathrm{ar}\, p}) \mid p \in \mathsf{p},\ t_1, \ldots, t_{\mathrm{ar}\, p} \in \mathsf{t} \,\}$ be the set of *atoms*.
Given a formal expression F, $\lceil F \rceil$ denotes the set of free variables in F.

For substitutions the usual notation and conventions apply. Let σ be the set of substitutions.

2.2 The FORUM$^{\parr\,\multimap\,\forall}$ *Presentation of a Fragment of Linear Logic*

Abstract logic programming has been defined in [12] as a very general way to define logic programming over any fragment of a sequent presentation of a logical system. In the case of linear logic, it has been recently discovered that the whole of linear logic may be seen as an abstract logic programming language [11]. In this work Miller shows a logical system, named FORUM, which naturally yields only uniform proofs, and, being this system correct and complete wrt linear logic, the underlying language is an abstract logic programming language. From our point of view this is important because having an abstract logic programming language naturally provides for sensible implementations. Moreover, in FORUM we can find a natural notion of state of a computation (the linear contexts in FORUM sequents). Purpose of this section is to present the fragment of FORUM necessary and sufficient to carry on our investigation, *viz.* to use the notion of state we have in order to represent causality and independence among agents. It should be noted that, holding for linear logic the cut-elimination property, any further enrichments of the fragment we present is possible, modularity among linear logic connectives being guaranteed by the cut-elimination property, and the validity of the consequent subformula property. The reader can find in [11] the details missing here. Methods are called this way after [13].

The set of *methods* M is the least set such that:

1) $\mathsf{A} \subset \mathsf{M}$.

2) If $M, M' \in \mathsf{M}$ then $(M \parr M') \in \mathsf{M}$ and $(M \multimap M') \in \mathsf{M}$.

3) If $M \in \mathsf{M}$ and $x \in \mathsf{x}$ then $(\forall x : M) \in \mathsf{M}$.

$\parr$ associates to the left and $\multimap$ associates to the right. Instead of $(\forall x_1 : (\ldots : (\forall x_h : M) \ldots))$ we shall write $(\forall x_1 \ldots x_h : M)$. Outermost parentheses shall be omitted whenever possible. If $h \leqslant k$ and $f \colon \mathsf{N}_h^k \to \mathsf{M}$, the notation $\parr_{i \in \mathsf{N}_h^k} f(i)$

stands for $f(h) \parr \cdots \parr f(k)$; given $g\colon \mathsf{N}_h^k \to \mathfrak{x}$, the notation $\bigparr_{i \in \mathsf{N}_h^k} g(i) : M$ stands for $\forall g(h) \ldots g(k) : M$. If $\bar{M} = (M|_1^h) \in \mathsf{M}^+$ then $\bigparr \bar{M}$ stands for $M_1 \parr \cdots \parr M_h$. If $\bar{x} = (x|_1^h) \in \mathfrak{x}^\star$ then $\bigparr \bar{x} : M$ stands for $\forall x_1 \ldots x_h : M$ when $h > 0$, and for M when $h = 0$.

We adopt a special kind of sequents, made up from collections of methods with different structures imposed on them: sets, multisets and sequences. Sets are used to represent information as in classical logic: this is information which does not change during the computation; a program is represented as a set of methods. Multisets are used to represent the state of the computation, which, of course, changes as the computation goes ahead; here is where linear logic has its main usefulness. Sequences of atoms appear in our sequents as a way to limit the choice in the use of right inference rules; this ordering does not affect correctness and completeness. From the proof theory point of view, sets are places where weakening and contraction rules are allowed, while on multisets and sequences these rules are forbidden. In these sequents there is room for one method (which we call "focused") which drives the choice of left inference rules.

The set of *sequents* Σ contains elements of the form $(\Psi; \Gamma | M \vdash \Lambda; \Xi)$, where $\Psi \in \mathsf{P_f}(\mathsf{M})$ (the *classical context*), Γ is a finite multiset of methods (the *left linear context*), $M \in \mathsf{M} \cup \{\epsilon_{\mathsf{M}}\}$ (the *focused method*), $\Lambda \in \mathsf{A}^\star$ (the *atomic context*) and $\Xi \in \mathsf{M}^\star$ (the *right linear context*). Instead of $(\Psi; \Gamma \mid \epsilon_{\mathsf{M}} \vdash \Lambda; \Xi)$ we shall write $(\Psi; \Gamma \vdash \Lambda; \Xi)$. In the following Ψ, Γ, Ξ and Λ shall stand for, respectively, sets, multisets and sequences of methods and sequences of atoms.

We outline a sequent presentation of a fragment of the FORUM inference system. FORUM imposes a discipline (wrt full linear logic) on the nondeterministic bottom-up construction of proofs, thereby drastically reducing their search space. It turns out that FORUM is equivalent to linear logic, but proofs in FORUM are uniform (see [12]). Since FORUM is much closer to the computations we are interested in, it greatly helped us in finding the way to relate them to linear logic.
The inference system we shall use as an intermediate step from SMR to linear logic is $\text{FORUM}^{\parr \multimap \forall}$, meaning that $\parr$, $\multimap$ and $\forall$ are the only logical connectives this subsystem of FORUM deals with.

Let $\text{FORUM}^{\parr \multimap \forall}$ be the set of inference rules obtained by the schemata in fig. 1. There, with $\Lambda \curlyvee \Lambda'$, we represent any sequence of atoms obtained by an ordered merge of Λ and Λ'. For example, $(A_1, A_2) \curlyvee (A_3, A_4)$ may stand for (A_1, A_3, A_2, A_4) or (A_3, A_1, A_2, A_4) or $\ldots$.

The link between $\text{FORUM}^{\parr \multimap \forall}$ and linear logic is established by the following proposition, which follows from the result in [11] and the cut-elimination

Structural rules

$$\mathsf{I}\,\frac{}{\Psi;\,|\,A\vdash A;}\qquad \mathsf{L}\,\frac{\Psi;\Gamma\vdash\Lambda,A;\Xi}{\Psi;\Gamma\vdash\Lambda;A,\Xi}\qquad \mathsf{D_L}\,\frac{\Psi;\Gamma\,|\,M\vdash\Lambda;}{\Psi;M,\Gamma\vdash\Lambda;}\qquad \mathsf{D_C}\,\frac{M,\Psi;\Gamma\,|\,M\vdash\Lambda;}{M,\Psi;\Gamma\vdash\Lambda;}$$

Left rules

$$\parr_{\mathsf{L}}\,\frac{\Psi;\Gamma\,|\,M\vdash\Lambda;\quad\Psi;\Gamma'\,|\,M'\vdash\Lambda';}{\Psi;\Gamma,\Gamma'\,|\,M\parr M'\vdash\Lambda\curlyvee\Lambda';}$$

$$\multimap_{\mathsf{L}}\,\frac{\Psi;\Gamma\vdash\Lambda;M\quad\Psi;\Gamma'\,|\,M'\vdash\Lambda';}{\Psi;\Gamma,\Gamma'\,|\,M\multimap M'\vdash\Lambda\curlyvee\Lambda';}$$

$$\forall_{\mathsf{L}}\,\frac{\Psi;\Gamma\,|\,M[t/x]\vdash\Lambda;}{\Psi;\Gamma\,|\,\forall x:M\vdash\Lambda;}$$

Right rules

$$\parr_{\mathsf{R}}\,\frac{\Psi;\Gamma\vdash\Lambda;M,M',\Xi}{\Psi;\Gamma\vdash\Lambda;M\parr M',\Xi}$$

$$\multimap_{\mathsf{R}}\,\frac{\Psi;M,\Gamma\vdash\Lambda;M',\Xi}{\Psi;\Gamma\vdash\Lambda;M\multimap M',\Xi}$$

$$\forall_{\mathsf{R}}\,\frac{\Psi;\Gamma\vdash\Lambda;M[x/x'],\Xi}{\Psi;\Gamma\vdash\Lambda;\forall x':M,\Xi}{}^{\star}$$

*where x is not free in the conclusion

FIG. 1 The FORUM$^{\parr\multimap\forall}$ fragment of FORUM

theorem.

2.2.1 THEOREM *A sequent* $(M|_1^h;\ \vdash\ ;M)$ *has a proof in* FORUM$^{\parr\multimap\forall}$ *iff* $(!M_1\multimap\cdots\multimap!M_h\multimap M)$ *has a proof in linear logic.*

3 Syntax and Operational Semantics of SMR

In this section we show how a natural notion of causality and independence among agents may be obtained in FORUM$^{\parr\multimap\forall}$. Agents are atoms among which a partial order is defined. Every agent can be reduced into a partial order of agents, or solved. In any case the structure of the context in which the reduction takes place is conserved. We define the language SMR as a natural way to define reduction of agents preserving a partial order. On this language computations are defined, and in the next section we shall show how SMR and its computations can be related, respectively, to FORUM and the (uniform) search for derivations in FORUM, respectively. SMR is in the logic programming style, of course, then we define goals and clauses, and the main computational mechanism is resolution, or, more appropriately, backchaining. For a more thorough presentation of SMR we refer the reader to [8].

We build up the language of goals starting from the empty goal $\circ$ and the set of atoms A, and freely composing with the two connectives $\diamond$ and

$\triangleleft$. The connectives have to be thought of as associative and non-idempotent operators; moreover, $\diamond$ is commutative and $\triangleleft$ is not. The empty goal $\circ$ behaves as a unity for $\diamond$ and $\triangleleft$, like *true* does for the classical logic connective *and*. In the translation from SMR into linear logic it shall be mapped to an atom of a special class.

Suppose, from now on, that a special 0-arity predicate $\circ$ is in A. We shall call $\circ$ the *empty goal*. Given a substitution σ, define $\circ\sigma = \circ$.

The set of *goals* G is the least set such that:

1) $\mathsf{A} \subset \mathsf{G}$.

2) If $G, G' \in \mathsf{G}$ then $(G \diamond G') \in \mathsf{G}$ and $(G \triangleleft G') \in \mathsf{G}$.

$\diamond$ and $\triangleleft$ are, respectively, the *parallel* and *sequential connective*; goals of the form $(G \diamond G')$ and $(G \triangleleft G')$ are, respectively, *parallel* and *sequential goals*. Application of substitutions extends naturally to goals.

Commutativity of $\diamond$ induces an equivalence relation on goals.

Two goals are equivalent if they only differ by the order of goals connected by $\diamond$ in parallel subgoals.
From now on we shall consider G as the set of equivalence classes induced by this equivalence relation.

The top of a goal is the multiset of atoms in the goal not preceded by other atoms; formally:

The function $\mathrm{top}\colon \mathsf{G} \to \mathsf{P}_+(\mathsf{G})$ is defined as

$$\mathrm{top}\, G = \begin{cases} G & \text{if } G \in \mathsf{A}, \\ \mathrm{top}\, G' \uplus \mathrm{top}\, G'' & \text{if } G = G' \diamond G'', \\ \mathrm{top}\, G' & \text{if } G = G' \triangleleft G''. \end{cases}$$

SMR consists of three components: a set of programs, the set of goals we already defined and a transition relation which models the nondeterministic transformation of goals into goals.
A program is a finite set of clauses. Each clause specifies the synchronous rewriting of some atoms in the top of a goal into the same number of goals. Rewriting takes place in the context of a larger goal, in which the rewritten atoms, considered as a multiset, are unifiable with the head of the clause, again considered as a multiset.
The clause specifies also which goal takes the place of which atom (matching one of the atoms in its head), and the usual logic programming mechanism of instantiation with the unifier takes place. We do not insist on the unifiers being mgu's, though this special case can easily be accommodated in our setting.

Let

$$\mathsf{D} = \{\, (A|_1^h \leftarrow G|_1^h) \mid h \in \mathsf{N}_1,\ A_1, \ldots, A_h \in \mathsf{A} \setminus \{\circ\} \,\}$$

be the set of (*distributed*) *clauses*.
Atoms $A_1, \ldots, A_h$ constitute the *head* of the clause; goals $G_1, \ldots, G_h$ constitute its *body*.
Application of substitutions is extended in the natural way to clauses.

Variables in clauses are considered as universally quantified. Then two clauses which only differ by a renaming of variables are to be considered the same.

Clearly, clauses establish a one-to-one correspondence between atoms in the head and goals in the body. We shall consider the same clauses which only differ by the order of atoms in the head and goals in the body, provided the correspondence is respected.

$\approx$ is the least equivalence relation on clauses such that:

1) $D \approx D'$ if $D = D'\rho$, where ρ is a renaming substitution for all variables in D;
2) $(A_1, \ldots, A_i, \ldots, A_j, \ldots, A_h \leftarrow G_1, \ldots, G_i, \ldots, G_j, \ldots, G_h) \approx$
   $(A_1, \ldots, A_j, \ldots, A_i, \ldots, A_h \leftarrow G_1, \ldots, G_j, \ldots, G_i, \ldots, G_h)$.

From now on we shall consider D as quotiented by the $\approx$ equivalence on clauses.

Let $\mathsf{P} = \mathsf{P}_{\mathsf{f}}(\mathsf{D})$ be the set of *programs*.

SMR operational semantics shall be given in the SOS style. We shall define a transition relation on configurations, a configuration being a program, a goal, and an optional selected clause which the goal is resolved by.

A *configuration* is either a triple $\langle P, D, G\rangle$ or a pair $\langle P, G\rangle$, where P is a program, D is a clause and G is a goal.

We need a notion of merge of two clauses:

Given two clauses $D = (A_1, \ldots, A_h \leftarrow G_1, \ldots, G_h)$ and $D' = A'_1, \ldots, A'_{h'} \leftarrow G'_1, \ldots, G'_{h'}$, let us define the *merge* of D and D' as

$$D \curlyvee D' = A_1, \ldots, A_h, A'_1, \ldots, A'_{h'} \leftarrow G_1, \ldots, G_h, G'_1, \ldots, G'_{h'}.$$

We are now ready to define the operational semantics of SMR. A more detailed discussion can be found in [8].

In fig. 2 a SOS-style operational semantics is defined; the rules define the binary relation $\rightarrow$ on configurations.
Given P, the *resolution* relation $\overset{}{\underset{P}{\rightharpoonup\!\diamond}} \subset \mathsf{G}^2 \times \sigma$ is such that $(G, G', \sigma) \in \overset{}{\underset{P}{\rightharpoonup\!\diamond}}$ (written $G \overset{\sigma}{\underset{P}{\rightharpoonup\!\diamond}} G'$) whenever $\langle P, G\rangle \rightarrow \langle P, D\sigma', G\sigma\rangle \rightarrow \langle P, G'\rangle$.

Selection of a clause

$$\frac{D \in P}{\langle P, G\rangle \to \langle P, D\rho\sigma', G\sigma\rangle}, \text{ where } \sigma \text{ and } \sigma' \text{ are substitutions and } \rho \text{ is a renaming substitution}$$

Atomic resolution

$$\frac{}{\langle P, A \leftarrow G, A\rangle \to \langle P, G\rangle}$$

Parallel resolution

$$\frac{\langle P, D, G\rangle \to \langle P, G'\rangle \quad \langle P, D', G''\rangle \to \langle P, G'''\rangle}{\langle P, D \curlyvee D', G \diamond G''\rangle \to \langle P, G' \diamond G'''\rangle} \qquad \frac{\langle P, D, G\rangle \to \langle P, G'\rangle}{\langle P, D, G \diamond G''\rangle \to \langle P, G' \diamond G''\rangle}$$

Sequential resolution

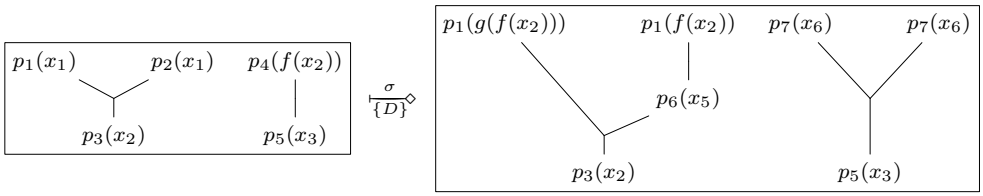
$$\frac{\langle P, D, G\rangle \to \langle P, G'\rangle}{\langle P, D, G \triangleleft G''\rangle \to \langle P, G' \triangleleft G''\rangle}$$

Reduction of empty goals

$$\frac{}{\langle P, G \diamond \circ\rangle \to \langle P, G\rangle} \qquad \frac{}{\langle P, G \triangleleft \circ\rangle \to \langle P, G\rangle} \qquad \frac{}{\langle P, \circ \triangleleft G\rangle \to \langle P, G\rangle}$$

$$\frac{\langle P, G\rangle \to \langle P, G'\rangle \quad \langle P, G''\rangle \to \langle P, G'''\rangle}{\langle P, G \diamond G''\rangle \to \langle P, G' \diamond G'''\rangle} \qquad \frac{\langle P, G\rangle \to \langle P, G'\rangle \quad \langle P, G''\rangle \to \langle P, G'''\rangle}{\langle P, G \triangleleft G''\rangle \to \langle P, G' \triangleleft G'''\rangle}$$

FIG. 2 SMR operational semantics



$D = (p_4(x_4), p_2(g(x_4)) \leftarrow p_7(x_3) \diamond p_7(x_3), p_1(x_4) \triangleleft p_6(x_5))$ and $\sigma = [g(f(x_2))/x_1]$

FIG. 3 Example of resolution

The *reduction* relation $\succ \subset \mathsf{G}^2$ is such that $G \succ G'$ whenever $\langle P, G\rangle \to \langle P, G'\rangle$. Define the relation $\underset{P}{\rightharpoonup}^{\diamond} \subset \mathsf{G}^2 \times \sigma$ as $\underset{P}{\rightharpoonup}^{\diamond} = \succ \times (\{\epsilon\} \cup \vdash\!\!\!\underset{P}{}^{\diamond})$. Instead of $(G, G', \sigma) \in \underset{P}{\rightharpoonup}^{\diamond}$ we shall write $G \overset{\sigma}{\underset{P}{\rightharpoonup}}^{\diamond} G'$.

Let SMR be the triple $(\mathsf{P}, \mathsf{G}, \{ \underset{P}{\rightharpoonup}^{\diamond} \mid P \in \mathsf{P} \})$.

$\vdash\!\!\!\underset{P}{}^{\diamond}$ and $\succ$ represent elementary steps in a computation in SMR. $\succ$ is

trivial; in fig. 3 an example of resolution is shown.

A transition $G_0 \overset{\sigma_1}{\underset{P}{\diamond}} \cdots \overset{\sigma_h}{\underset{P}{\diamond}} G_h$ is a G-*computation* (*by* P); if $G_h = \circ$ it is a *successful* G-computation *of* G_0 *yielding* $\sigma_1 \cdots \sigma_h$.
Define the relation $\overset{}{\underset{P}{\Diamond}} \subset \mathsf{G}^2 \times \mathsf{\sigma}$ as $\overset{}{\underset{P}{\Diamond}} = \{ (G_0, G_h, \sigma_1 \cdots \sigma_h) \mid G_0 \overset{\sigma_1}{\underset{P}{\diamond}} \cdots \overset{\sigma_h}{\underset{P}{\diamond}} G_h \}$. Instead of $(G, G', \sigma) \in \overset{}{\underset{P}{\Diamond}}$ we shall write $G \overset{\sigma}{\underset{P}{\Diamond}} G'$.

4 SMR and Linear Logic

We shall now briefly present a translation of SMR goals and clauses into linear logic. In fact, the translation is into FORUM$^{\parr \multimap \forall}$, and we claim that G-computations are represented by FORUM$^{\parr \multimap \forall}$ derivations in a correct and complete way. Then, in the end, a G-computation is no more and no less than the proof of a suitable formula in linear logic.
More details may be found in [8].

Let us augment the set of variables by a denumerable set π of *process variables*, which are not allowed to appear in SMR atoms. SMR atoms are translated into atoms. The terms inside are left untouched, and the relative position in the goal (coordinate) yields a unique process variable, which is appended to the resulting atom. Since atoms in SMR do not contain process variables, name clashes are avoided. The empty goal translates into a special atom of the kind $\circ(\pi)$.

Let $\circ$ be a distinguished predicate of arity 1. The function $\mathsf{p}[\![\cdot]\!] \colon \mathsf{p} \to \mathsf{p}$ is chosen such that it is one-one, it holds $\operatorname{ar} \mathsf{p}[\![p]\!] = \operatorname{ar} p + 1$ and $\mathsf{p}[\![\circ]\!] = \circ$.

Let $\kappa = \mathsf{N}^\star$ be the set of *coordinates*.

Coordinates shall provide a convenient way to generate distinct process variables.

While π stands for a generic process variable, object process variables are ϕ_{κ_1}, ϕ_{κ_2}, ..., where the indexes are coordinates.

Define $\mathsf{A}[\![\cdot]\!] \colon \mathsf{A} \times \kappa \to \mathsf{A}$ as $\mathsf{A}[\![p(t|_1^h), \kappa]\!] = \mathsf{p}[\![p]\!](t|_1^h, \phi_\kappa)$. We shall write $A\phi_\kappa$ instead of $\mathsf{A}[\![A, \kappa]\!]$.
We shall refer to atoms $\circ\pi$ as *success atoms*.

In the following translation the structure of the goal is kept by process variables (identity of atoms in the structure), by the $\parr$ connective (parallelism among atoms) and by the $\multimap$ connectives (directionality of sequential connectives).

For every κ define $\mathsf{G}[\![\cdot]\!]_\kappa \colon \mathsf{G} \to \mathsf{M}$ as

$$\begin{aligned}
\mathsf{G}[\![A]\!]_\kappa &= \mathsf{A}[\![A, \kappa]\!] = A\phi_\kappa, \\
\mathsf{G}[\![G \diamond G']\!]_\kappa &= \mathsf{G}[\![G]\!]_{\kappa\|1} \parr \mathsf{G}[\![G']\!]_{\kappa\|2}, \\
\mathsf{G}[\![G \triangleleft G']\!]_\kappa &= (\mathsf{G}[\![G']\!]_{\kappa\|2} \multimap \mathsf{B}[\![G]\!]_{\kappa\|1}) \multimap \mathsf{G}[\![G]\!]_{\kappa\|1},
\end{aligned}$$

where

$$\begin{aligned}
\mathsf{B}[\![A]\!]_\kappa &= \mathsf{A}[\![\circ, \kappa]\!] = \circ\phi_\kappa, \\
\mathsf{B}[\![G \diamond G']\!]_\kappa &= \mathsf{B}[\![G]\!]_{\kappa\|1} \parr \mathsf{B}[\![G']\!]_{\kappa\|2}, \\
\mathsf{B}[\![G \triangleleft G']\!]_\kappa &= \mathsf{B}[\![G']\!]_{\kappa\|2}.
\end{aligned}$$

Define $\mathsf{D}[\![\cdot]\!] \colon \mathsf{D} \to \mathsf{M}$ as

$$\mathsf{D}[\![A|_1^h \leftarrow G|_1^h]\!] = \bigvee \lceil A|_1^h \leftarrow G|_1^h \rceil : \bigvee_{i \in \mathsf{N}_1^h} \phi_i : (\parr_{i \in \mathsf{N}_1^h} \mathsf{D}[\![G_i]\!]'_i \multimap \parr_{i \in \mathsf{N}_1^h} A_i\phi_i),$$

where, for every κ, $\mathsf{D}[\![\cdot]\!]'_\kappa \colon \mathsf{G} \to \mathsf{M}$ is defined as

$$\mathsf{D}[\![G]\!]'_\kappa = \begin{cases} G\phi_\kappa & \text{if } G \in \mathsf{A} \\ \bigvee V : ((\circ\phi_\kappa \multimap \mathsf{B}[\![G]\!]_\kappa) \multimap \mathsf{G}[\![G]\!]_\kappa) & \text{if } G \notin \mathsf{A} \end{cases},$$

where V is the set of process variables appearing in $\mathsf{G}[\![G]\!]_\kappa$.

We are now ready for the main result of this section.

Given a program P and a goal G, let the following sequent be a *representation* for configuration $\langle P, G\rangle$:

$$\mathsf{D}[\![P]\!]; \vdash \mathsf{G}[\![G]\!]_\epsilon.$$

A correctness and completeness theorem may be stated, which can be chained to the correctness and completeness result between FORUM$^{\parr \multimap \forall}$ and linear logic, given at the end of section 2.

4.1 THEOREM *A successful* G*-computation by P of G yielding σ exists iff there is a proof in* FORUM$^{\parr \multimap \forall}$ *of the representation of $\langle P, G\sigma\rangle$.*

5 NABLA Syntax and Operational Semantics

The first subsection deals with basic programs, the second one with their composition. Differences between NABLA and GAMMA are highlighted, but a discussion about them is postponed to the final conclusions.

Basic programs recursively rewrite a multiset of elements of a certain universe. We suppose elements of this universe are already terms in the syntax which SMR and FORUM are built upon. This restriction is not actually limiting, since their replacement with more general structures is allowed by the treatment that follows.

Let c be a denumerable set of *constants* and suppose c is a subset of the set of constants in SMR.

Multiset rewriting occurs (or stops) when some relation is satisfied. We are interested in decidable relations, and we shall suppose to have at hand an algorithm, expressed in some language, which decides relations over every h-uple of constants, when necessary.

For $h \in \mathsf{N}$, let $\delta^h = \{\, \delta \subseteq \mathsf{c}^h \mid \delta \text{ is decidable} \,\}$. Let $\delta = \bigcup_{i \in \mathsf{N}} \delta^i$ be the set of *reactions*.

Depending on the satisfaction of some relation in δ, an action is performed. Actions rewrite multisets of constants, and we require that they are total and computable functions.

For $h, k \in \mathsf{N}$, define $\gamma^{h,k} = \{\, \gamma \colon \mathsf{c}^h \to \mathsf{c}^k \mid \gamma \text{ is total and computable} \,\}$. Let $\gamma = \bigcup_{i,j \in \mathsf{N}} \gamma^{i,j}$ be the set of *actions*.

Reactions and actions come together in pairs. Given a multiset M of constants of c, we consider multiset rewritings of the kind $M \xrightarrow{(\delta,\gamma)} M'$: such a step can be performed if there exists $(c|_1^h)$ such that $\{c|_1^h\}_+ \Subset M$ and $(c|_1^h) \in \delta$; then $M' = (M \setminus_+ \{c|_1^h\}_+) \uplus \{c'|_1^k\}_+$, where $(c'|_1^k) = \gamma(c|_1^h)$.

For every $h, k \in \mathsf{N}$ and for every $\delta \in \delta^h$ and $\gamma \in \gamma^{h,k}$ define the relation $\xrightarrow{(\delta,\gamma)}$ as the set $\big\{\, (M, M') \;\big|\; M, M' \colon \mathsf{c} \to \mathsf{N},\ \exists (c|_1^h) \in \delta : \big(\{c|_1^h\}_+ \Subset M \wedge (c'|_1^k) = \gamma(c|_1^h) \wedge M' = (M \setminus_+ \{c|_1^h\}_+) \uplus \{c'|_1^k\}_+\big) \,\big\}$.

A basic program in GAMMA is a pair (δ, γ), and a configuration is a pair $\langle (\delta, \gamma), M \rangle$, where M is a multiset of constants of c. We say that a configuration is terminating, and we write $\langle (\delta, \gamma), M \rangle \downarrow^{\mathsf{r}}$, if (δ, γ) can not produce any further computation step from M. So, this is the operational semantics of a single computation step of GAMMA programs, in SOS style:

$$\frac{M \xrightarrow{(\delta,\gamma)} M'}{\langle (\delta, \gamma), M \rangle \to_{\mathsf{r}} \langle (\delta, \gamma), M' \rangle}, \qquad \frac{\neg \exists M' : (M, M') \in \xrightarrow{(\delta,\gamma)}}{\langle (\delta, \gamma), M \rangle \downarrow^{\mathsf{r}}}.$$

Then a computation in GAMMA terminates whenever the reaction is no more applicable.

For very general reasons, discussed elsewhere in this paper, we are not able to deal with this kind of termination in the "proof search as computation" paradigm in linear logic. To be successful we need to make more explicit the termination condition. Our position is the following: a program *can* terminate if some (terminating) reaction is applicable. Then we change the notion of basic program: the resulting language is called NABLA.

The set of *basic programs* B is defined as

$$\{\, (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}}) \mid \exists h_{\mathsf{N}}, k_{\mathsf{N}}, h_{\mathsf{T}}, k_{\mathsf{T}} \in \mathsf{N} : (\delta_{\mathsf{N}} \in \delta^{h_{\mathsf{N}}} \wedge \gamma_{\mathsf{N}} \in \gamma^{h_{\mathsf{N}}, k_{\mathsf{N}}} \wedge \delta_{\mathsf{T}} \in \delta^{h_{\mathsf{T}}} \wedge \gamma_{\mathsf{T}} \in \gamma^{h_{\mathsf{T}}, k_{\mathsf{T}}}) \,\}.$$

A *basic configuration* is a triple $\langle B, M \rangle_\theta$, where $B \in \mathsf{B}$, M is a multiset $M \colon \mathsf{c} \to \mathsf{N}$ and $\theta \in \{\mathsf{N}, \mathsf{T}\}$. A basic configuration $\langle B, M \rangle_{\mathsf{N}}$ is said *non-terminating*, whereas $\langle B, M \rangle_{\mathsf{T}}$ is said *terminating*.

The binary relation $\to_\nabla$ on basic configurations is the least one satisfying:

$$\mathsf{b}_{\mathsf{N}} \frac{M \xrightarrow{(\delta_{\mathsf{N}}, \gamma_{\mathsf{N}})} M'}{\langle (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}}), M \rangle_{\mathsf{N}} \to_\nabla \langle (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}}), M' \rangle_{\mathsf{N}}},$$

$$\mathsf{b}_{\mathsf{T}} \frac{M \xrightarrow{(\delta_{\mathsf{T}}, \gamma_{\mathsf{T}})} M'}{\langle (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}}), M \rangle_{\mathsf{N}} \to_\nabla \langle (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}}), M' \rangle_{\mathsf{T}}}.$$

In this way a computation stops after a certain relation is satisfied, whereas in GAMMA this only happens when a relation can not be satisfied.

5.2 Parallel and Sequential Composition of Basic Programs

We now introduce two composition operators on NABLA programs, *viz.* parallel and sequential composition. Again, this is similar to what has been done for GAMMA.

The set of *programs* Q is the least set such that:

1) $\mathsf{B} \subset \mathsf{Q}$.

2) If $Q, Q' \in \mathsf{Q}$ then $(Q \mid Q') \in \mathsf{Q}$ and $(Q \,;\, Q') \in \mathsf{Q}$.

$(Q \mid Q')$ is a *parallel* program, $(Q \,;\, Q')$ is a *sequential* one. A *configuration* is a triple $\langle Q, M \rangle_\theta$, where $Q \in \mathsf{Q}$, M is a multiset $M \colon \mathsf{c} \to \mathsf{N}$ and θ can be N, if $Q \notin \mathsf{B}$, or, if $Q \in \mathsf{B}$, can be either N or T. $\langle Q, M \rangle_{\mathsf{N}}$ and $\langle Q, M \rangle_{\mathsf{T}}$ are said, respectively, *non-terminating* and *terminating*.

$$\mathsf{p}_{\mathsf{NL}}\ \frac{\langle Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q', M'\rangle_{\mathsf{N}}}{\langle Q \mid Q'', M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q' \mid Q'', M'\rangle_{\mathsf{N}}} \qquad \mathsf{p}_{\mathsf{TL}}\ \frac{\langle Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q, M'\rangle_{\mathsf{T}}}{\langle Q \mid Q', M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q', M'\rangle_{\mathsf{N}}}$$

$$\mathsf{p}_{\mathsf{NR}}\ \frac{\langle Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q', M'\rangle_{\mathsf{N}}}{\langle Q'' \mid Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q'' \mid Q', M'\rangle_{\mathsf{N}}} \qquad \mathsf{p}_{\mathsf{TR}}\ \frac{\langle Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q, M'\rangle_{\mathsf{T}}}{\langle Q' \mid Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q', M'\rangle_{\mathsf{N}}}$$

$$\mathsf{s}_{\mathsf{N}}\ \frac{\langle Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q', M'\rangle_{\mathsf{N}}}{\langle Q\,;\,Q'', M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q'\,;\,Q'', M'\rangle_{\mathsf{N}}} \qquad \mathsf{s}_{\mathsf{T}}\ \frac{\langle Q, M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q, M'\rangle_{\mathsf{T}}}{\langle Q\,;\,Q', M\rangle_{\mathsf{N}} \to_{\nabla} \langle Q', M'\rangle_{\mathsf{N}}}$$

FIG. 4 SOS operational semantics of composed NABLA programs

The binary relation $\to_{\nabla}$ on configurations extends $\to_{\nabla}$ previously defined over basic configurations and, moreover, is the least one satisfying the SOS rules in fig. 4.

Let $\twoheadrightarrow_{\nabla}$ be the transitive and reflexive closure of $\to_{\nabla}$. If $\langle Q, M\rangle_{\mathsf{N}} \twoheadrightarrow_{\nabla} \langle B, M'\rangle_{\mathsf{T}}$ we say that there exists a successful (∇-)*computation* of M' from $\langle Q, M\rangle_{\mathsf{N}}$, and we write $\langle Q, M\rangle_{\mathsf{N}} \twoheadrightarrow_{\nabla} M'$.

An important difference between NABLA operational semantics and the GAMMA one lies in the treatment of parallel composition: it is asynchronous in NABLA and synchronous in GAMMA. The only parallel termination rule in GAMMA, in fact, is

$$\frac{\langle Q, M\rangle{\downarrow^{\mathsf{r}}} \quad \langle Q', M\rangle{\downarrow^{\mathsf{r}}}}{\langle Q \mid Q', M\rangle{\downarrow^{\mathsf{r}}}}.$$

Our weaker notion of termination provides for a straightforward specification in SMR and, hence, in linear logic.

6 NABLA and SMR

We first show that we can use SMR to decide relations in δ and to evaluate expressions in γ. An encoding of Turing machines into SMR would suffice. We shall stem from the more intuitive fact that SMR generalizes Horn clauses, for which a Turing computability result is already known. The following two propositions hold.

6.1 PROPOSITION *Given $\delta \in \delta^h$ there exist $P_\delta \in \mathsf{P}$ and $p_\delta \in \mathsf{p}$, with $\operatorname{ar} p_\delta = h$, such that for every $(c|_1^h) \in \mathsf{c}^h$ there exists a successful G-computation of $p_\delta(c|_1^h)$ iff $(c|_1^h) \in \delta$.*

SKETCH OF PROOF Since with Horn clauses we can compute every computable function, we only need to show that SMR generalizes Horn clauses wrt operational semantics. To see this, transform every Horn goal $A_1, \ldots, A_k$ into an SMR goal $A_1 \diamond \cdots \diamond A_k$ and every Horn clause $A_0 \leftarrow A_1, \ldots, A_k$ into an SMR clause $A_0 \leftarrow A_1 \diamond \cdots \diamond A_k$.
It is trivial to see that to every successful computation on Horn clauses corresponds a successful SMR computation whose last steps are reduction steps which remove empty goals.
The converse is trickier. Intuitively, given an SMR computation, one has to delay all reduction steps towards the end (which is trivial since all the goals remain "flat"). Then one has to show that the resolution part of the SMR computation can naturally yield a Horn computation where all unifiers produced are lifted to the corresponding mgu's.

6.2 PROPOSITION *Given* $\gamma \in \boldsymbol{\gamma}^{h,k}$ *there exist* $P_\gamma \in \mathsf{P}$ *and* $p_\gamma \in \mathsf{p}$, *with* $\operatorname{ar} p_\gamma = h + k$, *such that for every* $(c|_1^h) \in \mathsf{c}^h$ *they hold*:

.1 *if* $(c'|_1^k) = \gamma(c|_1^h)$ *then there exists a successful* G*-computation of* $p_\gamma(c|_1^h, x|_1^k)$ *by* P_γ *yielding* σ *such that* $\sigma|_{\{x|_1^k\}} = [c'_1/x_1, \ldots, c'_k/x_k]$;

.2 *for every successful* G*-computation of* $p_\gamma(c|_1^h, x|_1^k)$ *by* P_γ *yielding* σ *it holds* $\sigma|_{\{x|_1^k\}} = [c'_1/x_1, \ldots, c'_k/x_k]$, *where* $(c'|_1^k) = \gamma(c|_1^h)$.

SKETCH OF PROOF The argument is similar to the previous proof's one.

We now show how to translate NABLA configurations into SMR programs and goals. First of all, we need to introduce some special syntactic objects.

Suppose having a distinguished unary predicate $\tilde{}$: instead of $\tilde{}(t)$ we shall write $\tilde{t}$. Given a constant c, we say that $\tilde{c}$ is its *reification*. Furthermore, suppose to every basic program B corresponds a distinguished 0-arity predicate $\dot{B}$. Let T be a distinguished atom. These newly defined syntactic objects only appear where explicitly shown.

Functions $[\![\cdot]\!]_{\mathsf{G}}$ and $[\![\cdot]\!]_{\mathsf{P}}$ on NABLA configurations and programs, with values in SMR goals and programs, respectively, are defined in figg. 5 and 6.
The choice of variables in $[\![B]\!]_{\mathsf{PN}}$ and $[\![B]\!]_{\mathsf{PT}}$ (fig. 6) is arbitrary, provided they all are pairwise distinct.
We suppose all programs P_{δ_θ} and P_{γ_θ} ($\theta \in \{\mathsf{N}, \mathsf{T}\}$) appearing in $[\![Q]\!]_{\mathsf{P}}$ are disjoint wrt to predicates which they are built upon (this does not diminish generality); in this way they are freely composable without side effects.

Given configuration $\langle Q, M\rangle_{\mathsf{N}}$, $[\![Q]\!]'_{\mathsf{G}}$ faithfully represents the ordering structure of Q in SMR goal $[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}}$; component basic programs are represented by the markings $\dot{B}$: they shall remain in the goal until the corresponding B is not terminated.

The only purpose of T is to allow rewriting in case of reactions and actions

$$
\begin{aligned}
[\![\langle Q, \{c|_1^h\}_+\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} &= [\![Q]\!]'_{\mathsf{G}} \diamond \mathsf{T} \diamond \tilde{c}_1 \diamond \cdots \diamond \tilde{c}_h \\
[\![\langle B, \{c|_1^h\}_+\rangle_{\mathsf{T}}]\!]_{\mathsf{G}} &= \mathsf{T} \diamond \tilde{c}_1 \diamond \cdots \diamond \tilde{c}_h \\
[\![Q \mid Q']\!]'_{\mathsf{G}} &= [\![Q]\!]'_{\mathsf{G}} \diamond [\![Q']\!]'_{\mathsf{G}} \\
[\![Q \,;\, Q']\!]'_{\mathsf{G}} &= [\![Q]\!]'_{\mathsf{G}} \triangleleft [\![Q']\!]'_{\mathsf{G}} \\
[\![B]\!]'_{\mathsf{G}} &= \dot{B}
\end{aligned}
$$

FIG. 5 Translation of configurations into SMR goals

$$
\begin{aligned}
&[\![Q \mid Q']\!]_{\mathsf{P}} = \\
&[\![Q \,;\, Q']\!]_{\mathsf{P}} = [\![Q]\!]_{\mathsf{P}} \cup [\![Q']\!]_{\mathsf{P}} \\
&\quad [\![B]\!]_{\mathsf{P}} = \{[\![B]\!]_{\mathsf{PN}}, [\![B]\!]_{\mathsf{PT}}\} \cup P_{\delta_{\mathsf{N}}} \cup P_{\gamma_{\mathsf{N}}} \cup P_{\delta_{\mathsf{T}}} \cup P_{\gamma_{\mathsf{T}}} \\
&\quad [\![B]\!]_{\mathsf{PN}} = \dot{B}, \mathsf{T}, \tilde{x}_1, \ldots, \tilde{x}_{h_{\mathsf{N}}} \leftarrow \dot{B}, \mathsf{T} \diamond p_{\delta_{\mathsf{N}}}(x|_1^{h_{\mathsf{N}}}) \diamond \big(p_{\gamma_{\mathsf{N}}}(x|_1^{h_{\mathsf{N}}}, x'|_1^{k_{\mathsf{N}}}) \triangleleft (\tilde{x}'_1 \diamond \cdots \diamond \tilde{x}'_{k_{\mathsf{N}}})\big), \underbrace{\circ, \ldots, \circ}_{h_{\mathsf{N}}} \\
&\quad [\![B]\!]_{\mathsf{PT}} = \dot{B}, \mathsf{T}, \tilde{x}_1, \ldots, \tilde{x}_{h_{\mathsf{T}}} \leftarrow \circ, \mathsf{T} \diamond p_{\delta_{\mathsf{T}}}(x|_1^{h_{\mathsf{T}}}) \diamond \big(p_{\gamma_{\mathsf{T}}}(x|_1^{h_{\mathsf{T}}}, x'|_1^{k_{\mathsf{T}}}) \triangleleft (\tilde{x}'_1 \diamond \cdots \diamond \tilde{x}'_{k_{\mathsf{T}}})\big), \underbrace{\circ, \ldots, \circ}_{h_{\mathsf{T}}}
\end{aligned}
$$

where $B = (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}})$, $\delta_{\mathsf{N}} \in \delta^{h_{\mathsf{N}}}$, $\gamma_{\mathsf{N}} \in \gamma^{h_{\mathsf{N}}, k_{\mathsf{N}}}$, $\delta_{\mathsf{T}} \in \delta^{h_{\mathsf{T}}}$, $\gamma_{\mathsf{T}} \in \gamma^{h_{\mathsf{T}}, k_{\mathsf{T}}}$,
$p_{\delta_{\mathsf{N}}}$, $P_{\delta_{\mathsf{N}}}$, $p_{\delta_{\mathsf{T}}}$, $P_{\delta_{\mathsf{T}}}$ satisfy prop. 6.1 and $p_{\gamma_{\mathsf{N}}}$, $P_{\gamma_{\mathsf{N}}}$, $p_{\gamma_{\mathsf{T}}}$, $P_{\gamma_{\mathsf{T}}}$ satisfy prop. 6.2

FIG. 6 Translation of NABLA programs into SMR programs

in δ^0 and $\gamma^{0,k}$.

So, a goal is a $\diamond$ composition of T, $[\![Q]\!]'_{\mathsf{G}}$ (for non-terminating configurations) and of all the reified constants in a corresponding configuration. The SMR program $[\![Q]\!]_{\mathsf{P}}$ is just the union of the clauses relative to all basic SOS rules deduced from Q: every basic program B in Q contributes with two main clauses ($[\![B]\!]_{\mathsf{PN}}$ and $[\![B]\!]_{\mathsf{PT}}$); the one for termination erases $\dot{B}$ from the goal when executed.
Please notice that the top of the goal is kept ground by allowing new constants to reach the top only *after* they are actually computed by some $p_{\gamma_\theta}(\cdots)$.

It should not be difficult to convince oneself that the proposed specification provides for a maximally parallel and asynchronous execution in SMR.

Correctness of the translation is stated this way, for a single step computation:

6.3 THEOREM *If* $\langle Q, M\rangle_{\mathsf{N}} \rightarrow_{\triangledown} \langle Q', M'\rangle_\theta$ *then*
$[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \overset{\sigma}{\underset{[\![Q]\!]_{\mathsf{P}}}{\Lsh}} [\![\langle Q', M'\rangle_\theta]\!]_{\mathsf{G}}$, *for all* $Q, Q' \in \mathsf{Q}$, $M, M' \colon \mathsf{c} \to \mathsf{N}$, $\theta \in \{\mathsf{N}, \mathsf{T}\}$ *and for some substitution* σ.

Generalizing this result to $\twoheadrightarrow_\nabla$ is straightforward:

6.4 THEOREM *If* $\langle Q, M\rangle_{\mathsf{N}} \twoheadrightarrow_\nabla \langle Q', M'\rangle_\theta$ *then* $[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \overset{\sigma}{\underset{[\![Q]\!]_{\mathsf{P}}}{\Lsh\!\Rsh}} [\![\langle Q', M'\rangle_\theta]\!]_{\mathsf{G}}$, *for all* $Q, Q' \in \mathsf{Q}$, $M, M' \colon \mathsf{c} \to \mathsf{N}$, $\theta \in \{\mathsf{N}, \mathsf{T}\}$ *and for some substitution* σ.

Completeness requires a different approach. We have to show that no matter how SMR (then, in the end, linear logic) computes on a given $[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}}$ by a program $[\![Q]\!]_{\mathsf{P}}$, it actually performs a ∇-computation. Of course, it has to be taken into account the fact that SMR carries a finer notion of computation step than NABLA does.

6.5 THEOREM *If* $[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \overset{\sigma}{\underset{[\![Q]\!]_{\mathsf{P}}}{\Lsh\!\Rsh}} [\![\langle Q', M'\rangle_\theta]\!]_{\mathsf{G}}$ *then* $\langle Q, M\rangle_{\mathsf{N}} \twoheadrightarrow_\nabla \langle Q', M'\rangle_\theta$, *for all* $Q, Q' \in \mathsf{Q}$, $M, M' \colon \mathsf{c} \to \mathsf{N}$, $\theta \in \{\mathsf{N}, \mathsf{T}\}$ *and* $\sigma \in \mathsf{\sigma}$.

Proofs of the correctness and completeness theorems are given in the appendix.

7 Conclusions

The first step on the way from linear logic to NABLA is the definition of FORUM. This step stems from very general considerations about logic programming [12]. It essentially provides a way to eliminate a great deal of non-determinism by resorting to a fragment of logic sound from the point of view of language design [11].

The second step, performed in this paper, is motivated by the need to have a minimum specification language with sequentiality, with strong grounds in first order linear logic. Here the key design analysis is about commutativity *vs.* non-commutativity of non-idempotent conjunctions. Of course the language must be expressive enough to allow recursion. This led us to SMR, and to the definition of the FORUM$^{\parr\,\multimap\forall}$ fragment.

We obtained both a declarative and operational understanding of sequencing by associating to every task a pair of statements: 1) that the task i has to be performed by an agent (say A_i) and 2) that when the task is accomplished a signal ($\circ_i$) is issued. The above treatment of sequentiality clearly encompasses paradigms more general than SMR. SMR by itself is a powerful language, as many examples show [9, 14].

The translation makes use of the full $\parr\,\multimap\forall$ fragment of linear logic, thus making full *logical* use of these connectives. This is opposed to, for exam-

ple, classical logic programming, in which $\Rightarrow$ and $\forall$ are only used in left rules. An important point is that all structural information in SMR goes into the logic, with no need to resort to trickeries with terms. We are also pleased by the correspondence between parts in the sequences of FORUM and our framework: the program in the classical context, the structure of the goal in the left linear context and the top of the goal in the atomic context. The translation is very conservative wrt computational complexity, and FORUM guarantees good operational properties.

The third step, from SMR to NABLA, is no more than an exercise in programming. SMR has enough expressive power to express every Turing equivalent algorithm, so it is adequate for basic programs, and has of course the ability to express a suitable sequential composition.

The key point here is in the design of NABLA itself. Two modifications to the GAMMA paradigm have been performed:

1) The termination condition for basic programs. Instead of terminating when *no* reactions are applicable, a program terminates when *some* termination reaction *is* applicable.
2) Synchronous termination of parallel programs is removed in favor of asynchronous one.

Both aspects have in common a *local* vs. *global* dichotomy. In 1, to terminate a GAMMA program we have to test on the *entire* multiset *every* possible reaction. We would need then a way to say, into the logic, that something is not provable. This is the same difficulty of giving proof-theoretical dignity to "negation as failure" in logic programming. Linear logic negation is, of course, a different thing. In other words, one has probably to change the logic, in ways not clear yet, to encompass such a "global" feature. As for the second aspect, the change is directly inspired by the nature of the $\parr$ connective. In the parallel "dimension" of the languages studied here there is really a smooth transition from $\parr$ to $\diamond$ to $|$. The proposed semantics for $|$, both the SOS-style and the SMR specification, are very natural and aesthetically appealing. Moreover, a synchronous termination again would require a global view of the multiset by the program. One can, of course, have this kind of global computations at the price of encoding the language into terms, which, by their nature, are entirely available for inspection by logical formulas. But this is contrary to the spirit this investigation has been inspired by, *i.e.* analyzing the logical content of GAMMA-like languages from the linear logic point of view.

The analysis presented represents a satisfying solution to the problem, given the constraints we imposed on our work: the use of linear logic, and in

particular its abstract logic programming presentation FORUM, the first-order restriction and the will not to use continuations.

Better solutions can possibly be found in the future. They will require more structural logics than linear logic, and logics able at the same time to better cope with globality. At present, non-commutative linear logic is not a feasible solution, since it has non-commutativity but lacks a natural notion of commutativity. Other approaches to globality, such as borrowing techniques from "negation as failure" frameworks, are destined to be only partial solutions since the beginning. In our opinion, pomset logic [15] shall provide an excellent framework for this kind of analysis, once we shall be able to define abstract logic programming in it. This is our active research field.

The problem, as should be clear by the previous exposition, is that while parallel composition (independence wrt the next step of computation) is very easy to deal with, the same is not true for sequential composition (causality). To deal with causality one has to make use of the implicit synchronization (*i.e.* causality!) obtainable by logic provability: one has to resort to linear implication. This fact is simply observable in the syntax of linear logic: while $\parr$ is commutative, $\multimap$ is not, and then we use $\multimap$.

This approach has at least one weak point: the partially ordered state is not directly represented. What is represented in linear contexts of FORUM fragments is a program which guarantees the exact dependencies of actions *over* the state. This is satisfactory from the operational point of view, because it guarantees all and only the correct computations. It is not satisfactory from the logical point of view because at this point one should expect to treat the structuring of state at a more direct level. In fact, in pomset logic, there are *two* connectives to use to build states: one commutative and the other non-commutative.

At the moment, a cut-elimination theorem for the sequent presentation of pomset logic is not proved, while it is proved for its proof nets. We hope that, when that theorem is finally proved, as we are convinced it is true, this work can immediately lead to a useful application of pomset logic. If not else, this kind of investigation is a challenging exercise to put on trial logical formalisms wrt the first class exploitation of state in concurrent computations.

Acknowledgments

We thank Paolo Ciancarini and Dale Miller for the many fruitful discussions.

Paola Bruscoli has been funded by ESPRIT BRA Project 9102 Coordination, Alessio Guglielmi by ESPRIT ParForCE (EP 6707).

Appendix

PROOF OF THEOREM 6.3 By induction on the definition of $\rightarrow_\nabla$ (*i.e.* on the structure of Q).
As a general remark, please notice the following fact. In $[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}}$ the subgoal $[\![Q]\!]'_{\mathsf{G}}$ is in parallel with T and the reified multiset, and is made up by unique basic program identifiers $\dot{B}$. In a G-computation the only modifications one can have on the initial $[\![Q]\!]'_{\mathsf{G}}$ are successive erasings of identifiers $\dot{B}$ in the top, until eventually it becomes $\circ$. A careful understanding of this is necessary in order to fill gaps in this proof, especially for the sequential cases.

1) Base case.
Rule b_{N}; let $Q = Q' = (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}})$ and $\theta = \mathsf{N}$. It holds $M \xrightarrow{(\delta_{\mathsf{N}}, \gamma_{\mathsf{N}})} M'$: let $(c|_1^{h_{\mathsf{N}}}) \in \delta_{\mathsf{N}}$ such that $M' = (M \setminus_+ \{c|_1^{h_{\mathsf{N}}}\}_+) \uplus \{c'|_1^{k_{\mathsf{N}}}\}_+$, where $(c'|_1^{k_{\mathsf{N}}}) = \gamma_{\mathsf{N}}(c|_1^{h_{\mathsf{N}}})$. Using propp. 6.1 and 6.2, and supposing $M = \{c|_1^{h_{\mathsf{N}}}\}_+ \uplus \{c''|_1^{h}\}_+$, where $h \in \mathsf{N}$, we have:

$$
\begin{aligned}
[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} &= \dot{Q} \diamond \mathsf{T} \diamond \tilde{c}_1 \diamond \cdots \diamond \tilde{c}_{h_{\mathsf{N}}} \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&\xmapsto[[\![Q]\!]_{\mathsf{P}}]{\sigma_1}\diamond\ \dot{Q} \diamond \mathsf{T} \diamond p_{\delta_{\mathsf{N}}}(c|_1^{h_{\mathsf{N}}}) \diamond \big(p_{\gamma_{\mathsf{N}}}(c|_1^{h_{\mathsf{N}}}, x'|_1^{k_{\mathsf{N}}}) \triangleleft (\tilde{x}'_1 \diamond \cdots \diamond \tilde{x}'_{k_{\mathsf{N}}})\big) \\
&\quad \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&\xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_2}\diamond\ \dot{Q} \diamond \mathsf{T} \diamond \big(p_{\gamma_{\mathsf{N}}}(c|_1^{h_{\mathsf{N}}}, x'|_1^{k_{\mathsf{N}}}) \triangleleft (\tilde{x}'_1 \diamond \cdots \diamond \tilde{x}'_{k_{\mathsf{N}}})\big) \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&\xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_3}\diamond\ \dot{Q} \diamond \mathsf{T} \diamond \tilde{c}'_1 \diamond \cdots \diamond \tilde{c}'_{k_{\mathsf{N}}} \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&= [\![\langle Q, M'\rangle_{\mathsf{N}}]\!]_{\mathsf{G}},
\end{aligned}
$$

where $\sigma = \sigma_1\sigma_2\sigma_3$ for some σ_1, σ_2 and σ_3.
Rule b_{T}; let $Q = Q' = (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}})$ and $\theta = \mathsf{T}$. It holds $M \xrightarrow{(\delta_{\mathsf{T}}, \gamma_{\mathsf{T}})} M'$: let $(c|_1^{h_{\mathsf{T}}}) \in \delta_{\mathsf{T}}$ such that $M' = (M \setminus_+ \{c|_1^{h_{\mathsf{T}}}\}_+) \uplus \{c'|_1^{k_{\mathsf{T}}}\}_+$, where $(c'|_1^{k_{\mathsf{T}}}) = \gamma_{\mathsf{T}}(c|_1^{h_{\mathsf{T}}})$. Using propp. 6.1 and 6.2, and supposing $M = \{c|_1^{h_{\mathsf{T}}}\}_+ \uplus \{c''|_1^{h}\}_+$, where $h \in \mathsf{N}$, we have:

$$
\begin{aligned}
[\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} &= \dot{Q} \diamond \mathsf{T} \diamond \tilde{c}_1 \diamond \cdots \diamond \tilde{c}_{h_{\mathsf{T}}} \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&\xmapsto[[\![Q]\!]_{\mathsf{P}}]{\sigma_1}\diamond\ \mathsf{T} \diamond p_{\delta_{\mathsf{T}}}(c|_1^{h_{\mathsf{T}}}) \diamond \big(p_{\gamma_{\mathsf{T}}}(c|_1^{h_{\mathsf{T}}}, x'|_1^{k_{\mathsf{T}}}) \triangleleft (\tilde{x}'_1 \diamond \cdots \diamond \tilde{x}'_{k_{\mathsf{T}}})\big) \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&\xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_2}\diamond\ \mathsf{T} \diamond \big(p_{\gamma_{\mathsf{T}}}(c|_1^{h_{\mathsf{T}}}, x'|_1^{k_{\mathsf{T}}}) \triangleleft (\tilde{x}'_1 \diamond \cdots \diamond \tilde{x}'_{k_{\mathsf{T}}})\big) \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&\xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_3}\diamond\ \mathsf{T} \diamond \tilde{c}'_1 \diamond \cdots \diamond \tilde{c}'_{k_{\mathsf{T}}} \diamond \tilde{c}''_1 \diamond \cdots \diamond \tilde{c}''_h \\
&= [\![\langle Q, M'\rangle_{\mathsf{T}}]\!]_{\mathsf{G}},
\end{aligned}
$$

where $\sigma = \sigma_1\sigma_2\sigma_3$ for some σ_1, σ_2 and σ_3.

2) Inductive case.
Rule p_{NL}; suppose $\langle Q \mid Q'', M\rangle_{\mathsf{N}} \to_\nabla \langle Q' \mid Q'', M'\rangle_{\mathsf{N}}$. We have:

$$[\![\langle Q \mid Q'', M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} = [\![Q'']\!]'_{\mathsf{G}} \diamond [\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \xrightarrow[[\![Q|Q''\!]\!]_{\mathsf{P}}]{\sigma}\!\!\diamond [\![Q'']\!]'_{\mathsf{G}} \diamond [\![\langle Q', M'\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} = [\![\langle Q' \mid Q'', M'\rangle_{\mathsf{N}}]\!]_{\mathsf{G}},$$

by induction hypothesis.
Rule p_{TL}; suppose $\langle Q \mid Q', M\rangle_{\mathsf{N}} \to_\nabla \langle Q', M'\rangle_{\mathsf{N}}$. We have:

$$[\![\langle Q \mid Q', M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} = [\![Q']\!]'_{\mathsf{G}} \diamond [\![\langle Q, M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \xrightarrow[[\![Q|Q'\!]\!]_{\mathsf{P}}]{\sigma}\!\!\diamond [\![Q']\!]'_{\mathsf{G}} \diamond [\![\langle Q, M'\rangle_{\mathsf{T}}]\!]_{\mathsf{G}} = [\![\langle Q', M'\rangle_{\mathsf{N}}]\!]_{\mathsf{G}},$$

by induction hypothesis.
Rules p_{NR} and p_{TR}: see above.
Rule s_{N}; suppose $\langle Q\,;Q'', M\rangle_{\mathsf{N}} \to_\nabla \langle Q'\,;Q'', M'\rangle_{\mathsf{N}}$. Supposing $M = \{c|_1^h\}_+$ and $M' = \{c'|_1^{h'}\}_+$, we have:

$$[\![\langle Q\,;Q'', M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} = ([\![Q]\!]'_{\mathsf{G}} \lhd [\![Q'']\!]'_{\mathsf{G}}) \diamond \mathsf{T} \diamond \tilde{c}_1 \diamond \cdots \diamond \tilde{c}_h \xrightarrow[[\![Q;Q''\!]\!]_{\mathsf{P}}]{\sigma}\!\!\diamond ([\![Q']\!]'_{\mathsf{G}} \lhd [\![Q'']\!]'_{\mathsf{G}}) \diamond \mathsf{T} \diamond \tilde{c}'_1 \diamond \cdots \diamond \tilde{c}'_{h'} = [\![\langle Q'\,;Q'', M'\rangle_{\mathsf{N}}]\!]_{\mathsf{G}},$$

by induction hypothesis.
Rule s_{T}; suppose $\langle Q\,;Q', M\rangle_{\mathsf{N}} \to_\nabla \langle Q', M'\rangle_{\mathsf{N}}$. Supposing $M = \{c|_1^h\}_+$ and $M' = \{c'|_1^{h'}\}_+$, we have:

$$\begin{aligned}[\![\langle Q\,;Q', M\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} = ([\![Q]\!]'_{\mathsf{G}} \lhd [\![Q']\!]'_{\mathsf{G}}) \diamond \mathsf{T} \diamond \tilde{c}_1 \diamond \cdots \diamond \tilde{c}_h \\ \xrightarrow[[\![Q;Q'\!]\!]_{\mathsf{P}}]{\sigma}\!\!\diamond (\circ \lhd [\![Q']\!]'_{\mathsf{G}}) \diamond \mathsf{T} \diamond \tilde{c}'_1 \diamond \cdots \diamond \tilde{c}'_{h'} \\ \succ_{\mathsf{G}} [\![Q']\!]'_{\mathsf{G}} \diamond \mathsf{T} \diamond \tilde{c}'_1 \diamond \cdots \diamond \tilde{c}'_{h'} = [\![\langle Q', M'\rangle_{\mathsf{N}}]\!]_{\mathsf{G}},\end{aligned}$$

by induction hypothesis.

PROOF OF THEOREM 6.5 The theorem is proved if, in the hypothesis, we can exhibit a G-computation

$$C = [\![\langle Q_0, M_0\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_0}\!\!\diamond [\![\langle Q_1, M_1\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_1}\!\!\diamond \cdots \xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_{k-1}}\!\!\diamond [\![\langle Q_k, M_k\rangle_\theta]\!]_{\mathsf{G}}$$

such that

$$\langle Q, M\rangle_{\mathsf{N}} = \langle Q_0, M_0\rangle_{\mathsf{N}} \to_\nabla \langle Q_1, M_1\rangle_{\mathsf{N}} \to_\nabla \cdots \to_\nabla \langle Q_k, M_k\rangle_\theta = \langle Q', M'\rangle_\theta.$$

This is true (by propp. 6.1 and 6.2) if, in C, every step

$$[\![\langle Q_i, M_i\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \xrightarrow[[\![Q]\!]_{\mathsf{P}}]{\sigma_i}\!\!\diamond [\![\langle Q_{i+1}, M_{i+1}\rangle_\theta]\!]_{\mathsf{G}}$$

takes one of these two forms, corresponding to basic configurations' SOS rules:

$$[\![\langle Q_i, M_i\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \overset{\sigma_i}{\underset{\{[\![B]\!]_{\mathsf{PN}}\}}{\mapsto\!\diamond}} G'_i \overset{\sigma'_i}{\underset{P_{\delta_{\mathsf{N}}}}{\ll\!\diamond}} G''_i \overset{\sigma''_i}{\underset{P_{\gamma_{\mathsf{N}}}}{\ll\!\diamond}} [\![\langle Q_{i+1}, M_{i+1}\rangle_{\mathsf{N}}]\!]_{\mathsf{G}}, \quad \text{where } Q_{i+1} = Q_i, \text{ or}$$

$$[\![\langle Q_i, M_i\rangle_{\mathsf{N}}]\!]_{\mathsf{G}} \overset{\sigma_i}{\underset{\{[\![B]\!]_{\mathsf{PT}}\}}{\mapsto\!\diamond}} G'_i \overset{\sigma'_i}{\underset{P_{\delta_{\mathsf{T}}}}{\ll\!\diamond}} G''_i \overset{\sigma''_i}{\underset{P_{\gamma_{\mathsf{T}}}}{\ll\!\diamond}} [\![\langle Q_{i+1}, M_{i+1}\rangle_{\theta}]\!]_{\mathsf{G}},$$

where $B = (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}})$ is a basic program appearing in Q_i, which disappears in Q_{i+1} if the second computation is performed.
The above fact can be proved by this argument:

1) Clauses in $[\![Q]\!]_{\mathsf{P}}$ are of two kinds: *main clauses* are all $[\![B]\!]_{\mathsf{PN}}$ and $[\![B]\!]_{\mathsf{PT}}$, for every basic program B appearing in Q; *auxiliary clauses* are clauses in *auxiliary programs* $P_{\delta_{\mathsf{N}}}$, $P_{\gamma_{\mathsf{N}}}$, $P_{\delta_{\mathsf{T}}}$ and $P_{\gamma_{\mathsf{T}}}$, for every basic program $B = (\delta_{\mathsf{N}}, \gamma_{\mathsf{N}}, \delta_{\mathsf{T}}, \gamma_{\mathsf{T}})$ appearing in Q. Let us call *auxiliary* both predicates and atoms appearing in auxiliary programs, too.
   Of course, *in every* $[\![\langle Q, M\rangle_{\theta}]\!]_{\mathsf{G}}$ *no auxiliary predicate appears.*
2) *If* $|C|_{\mathsf{R}} > 0$ *then there is at least one sub-computation of* C *of the kind* $p_{\delta_\theta}(\cdots) \overset{\sigma}{\underset{P_{\delta_\theta}}{\ll\!\diamond}} \circ$ *and one of the kind* $p_{\gamma_\theta}(\cdots) \overset{\sigma}{\underset{P_{\gamma_\theta}}{\ll\!\diamond}} \circ$. It is so because if $|C|_{\mathsf{R}} > 0$ there is at least one resolution with a main clause, and because every auxiliary atom thus introduced in the goal must eventually disappear, by point 1 above.
3) Given two computation steps $C_1 = G \overset{\sigma_1}{\underset{P_1}{\diamond}}_{\mathsf{G}} G'$ and $C_2 = G' \overset{\sigma_2}{\underset{P_2}{\diamond}}_{\mathsf{G}} G''$, we say that C_2 *switches with* C_1 if there exists G''' such that $G \overset{\sigma_2}{\underset{P_2}{\diamond}}_{\mathsf{G}} G''' \overset{\sigma_1}{\underset{P_1}{\diamond}}_{\mathsf{G}} G''$ and $\sigma_1\sigma_2 = \sigma_2\sigma_1$.
   In our case, it is not difficult (using programs' disjointness over auxiliary predicates) to prove that:
   Every step of the kind $G \overset{\sigma}{\underset{P}{\diamond}}_{\mathsf{G}} G'$, *where* P *is some auxiliary program, switches with every other step in a computation, except with*:
   - *those steps which are resolutions with clauses in the same auxiliary program* P;
   - *a resolution step with a main clause which introduces a predicate in* P.
4) Using 2 and 3, and proceeding by induction over its length, we obtain from every G-computation a G-computation like C above, which proves the theorem.

References

[1] Jean-Pierre Banâtre and Daniel Le Métayer. The Gamma model and its discipline of programming. *Science of Computer Programming*, 15(1):55–77, November 1990.

[2] C. Hankin, D. Le Métayer, and D. Sands. A calculus of Gamma programs. In *Languages and Compilers for Parallel Computing, 5th International Workshop*. Springer-Verlag, 1992.

[3] David Sands. Composed reduction systems. In *Proceedings of the 6th Nordic Workshop on Programming Theory*, number NS-94-6 in BRICS Notes Series, pages 360–377, Aarhus, Denmark, 1994.

[4] Paolo Ciancarini, Roberto Gorrieri, and Gianluigi Zavattaro. An alternative semantics for the calculus of Gamma programs. manuscript, 1995.

[5] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[6] Jean-Marc Andreoli and Remo Pareschi. Linear Objects: Logical processes with built-in inheritance. *New Generation Computing*, 9:445–473, 1991.

[7] Dale Miller. The π-calculus as a theory in linear logic: Preliminary results. In E. Lamma and P. Mello, editors, *1992 Workshop on Extensions to Logic Programming*, volume 660 of *Lecture Notes in Computer Science*, pages 242–265. Springer-Verlag, 1993.

[8] Alessio Guglielmi. Sequentiality by linear implication and universal quantification. In Jörg Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing, pages 160–174. Springer-Verlag, 1995.

[9] Luís Monteiro. Distributed logic: A logical system for specifying concurrency. Technical Report CIUNL-5/81, Departamento de Informática, Universidade Nova de Lisboa, 1981.

[10] Luís Monteiro. Distributed logic: A theory of distributed programming in logic. Technical report, Departamento de Informática, Universidade Nova de Lisboa, 1986.

[11] Dale Miller. A multiple-conclusion meta-logic. In S. Abramsky, editor, *Ninth Annual IEEE Symposium on Logic in Computer Science*, pages 272–281, Paris, July 1994.

[12] Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51:125–157, 1991.

[13] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[14] Alessio Guglielmi. Concurrency and plan generation in a logic programming language with a sequential operator. In P. Van Hentenryck, editor, *Logic Programming, 11th International Conference, S. Margherita Ligure, Italy*, pages 240–254. The MIT Press, 1994.

[15] Christian Retoré. Pomset logic: A non-commutative extension of classical linear logic. In *Computer Science Logic*, Paderborn, September 1995.